LAB 2 : (1-1-24)

WRITE a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operator (+) plus, (-) minus (*) multiply, (/) divide and ^ power

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
char st[MAX];
int top = -1;
void push (char st[], char);
char pop (char st[]);
void Infix to Postfix (char source[], char target[])
int getpri (char);

void main()
{
    char infix[100], postfix[100];
    printf ("Enter any infix expression \n");
    gets (infix);
    strcpy (postfix, "");
    Infix to Postfix (infix, postfix);
    printf ("The corresponding postfix expression is : \n");
    puts (postfix);
}

void  Infix to Postfix (char source[], char target[])
{
    int i=0, j=0;
```

```
char temp;
strcpy(target, "");
while(source[i] != '\0')
{
  if(source[i] == '(')
  {
    push(st, source[i]);
    i++;
  }
  else if(source[i] == ')')
  {
    while((top != -1) && (st[top] != '('))
    {
      target[j] = pop(st);
      j++;
    }
    if(top == -1)
    {
      printf("In Incorrect Expression");
      exit();
    }
    temp = pop(st);
    i++;
  }
  else if(isdigit(source[i]) || isalpha(source[i]))
  {
    target[j] = source[i];
    j++;
    i++;
  }
  else if(source[i] == '+' || source[i] == '-' || source[i]
```

```
    || source[i] == '/' || source[i] == '^')
{
    while ((top != -1) && (st[top] != '(') &&
           (getpri(st[top]) >= (getpri(source[i]))
    {
        target[j] = pop(st);
        j++;
    }
    push(st, source[i]);
    i++;
}
else
{
    printf("\n Incorrect element in expression");
    exit(1);
}
}
while((top != -1) && (st[top] != '('))
{
    target[j] = pop(st);
    j++;
}
target[j] = '\0';
}

int getpri(char op)
{
    if(op == '$')
        return 2;
    else if(op == '/' || op == '*')
```

```
return 1;
else if (op == '+' || op == '-')
    return 0;
}
void push (char st[], char val)
{
    if (top == MAX-1)
        printf ("\n Stack Overflow");
    else
    {
        top++;
        st[top] = val;
    }
}
char pop (char st[])
{
    Char val = ' ';
    if (top == -1)
        printf ("\n Stack underflow");
    else
    {
        val = st[top];
        top--;
    }
    return val;
}
```

Output:

Enter any infix expression

(A-B) * (C+D)

(AB-CD+)*